

Methods:

# Altering Classes, Methods & Variables

*In this lecture, we will go through some methods from Standard Ruby Library.  
You will learn how to change classes, modules, methods and all kinds of variables.*

# How can we change classes?

- Can change their methods & variables
- You can also “extend” them

# Extending Classes / Modules

```
module MyModule
  def foo
    "MyModule#foo"
  end

  def bar
    "MyModule#bar"
  end
end
```

```
class MyClass
  def foo
    "MyClass#foo"
  end
end
```

```
class MyClass
  include MyModule
  extend MyModule
end
```

```
obj = MyClass.new
obj.foo # MyClass#foo
```

```
obj.extend(MyModule)
obj.foo # MyModule#foo
obj.singleton_methods # [:foo, :bar]
```

```
obj2 = MyClass.new
obj2.foo # MyClass#foo
```

```
MyClass.extend MyModule
MyClass.singleton_class.instance_methods.include? :bar # true
```

# Altering existing Methods

```
class MyClass
  def foo
    "original foo"
  end
end
```

```
obj = MyClass.new
obj.foo # "original foo"
```

```
class MyClass
  def foo
    "changed foo"
  end
end
```

```
obj.foo # "changed foo"
```

# Altering Methods: `alias_method`, `alias`

```
# From ruby documentation:  
# http://ruby-doc.org/core/Module.html#method-i-alias\_method
```

```
module Mod  
  alias_method :orig_exit, :exit  
  def exit(code=0)  
    puts "Exiting with code #{code}"  
    orig_exit(code)  
  end  
end  
  
include Mod  
exit(99)                                # Exiting with code 99
```

# Altering Methods: `remove_method`, `undef_method`

```
class Parent
  def hello
    puts "In parent"
  end
end
```

```
class Child < Parent
  def hello
    puts "In child"
  end
end
```

```
c = Child.new
c.hello          # In child
```

```
class Child
  remove_method :hello
end
c.hello          # remove from child, still in parent
                # In parent
```

```
class Child
  undef_method :hello
end
c.hello          # prevent any calls to 'hello'
                # NoMethodError: undefined method `hello' for #<Child:0x401b3bb4>
```

# How can we change method visibility?

- private
- protected
- public
- private\_class\_method
- public\_class\_method

# Altering Methods: Visibility (public, private, etc.)

```
class MyClass
  def foo
  end
end
```

```
class MyClass
  private :foo
  protected :foo
  public :foo
```

```
# Method `foo` is now private
# Method `foo` is now protected
# Method `foo` is now public again
```

```
  private_class_method :new
  public_class_method :new
end
```

```
# Now, method `new` is private. You can't do `MyClass.new` anymore!
# Now, method `new` is public again!
```

```
class << MyClass
  private :new
end
```

```
# Same as saying: `private_class_method :new`
```



# How can we alter variables?

- Change constant visibility
- Create or set new value
- Remove them completely

# Altering Vars: Visibility (public\_constant, private\_constant)

```
class A
  class B
  end
end
```

```
obj = A::B.new # Can create. No problem!
```

```
class A
  private_constant :B
end
```

```
obj = A::B.new # NameError: private constant A::B referenced
```

# How can we change variable values?

- class\_variable\_set
- instance\_variable\_set
- const\_set
- local\_variable\_set

# Altering Variable Values (const\_set, etc.)

```
# From ruby documentation:  
# http://ruby-doc.org/core/Object.html#method-i-instance\_variable\_set
```

```
class Fred  
  def initialize(p1, p2)  
    @a, @b = p1, p2  
  end  
end
```

```
fred = Fred.new('cat', 99)
```

```
fred.instance_variable_set(:@a, 'dog')      # "dog"
```

```
fred.instance_variable_set(:@c, 'cat')      # "cat"
```

```
fred.inspect                                # #<Fred:0x401b3da8 @a=\"dog\", @b=99, @c=\"cat\">
```

# How can we remove variables?

- `remove_class_variable`
- `remove_instance_variable`
- `remove_const`

# Remove Variables

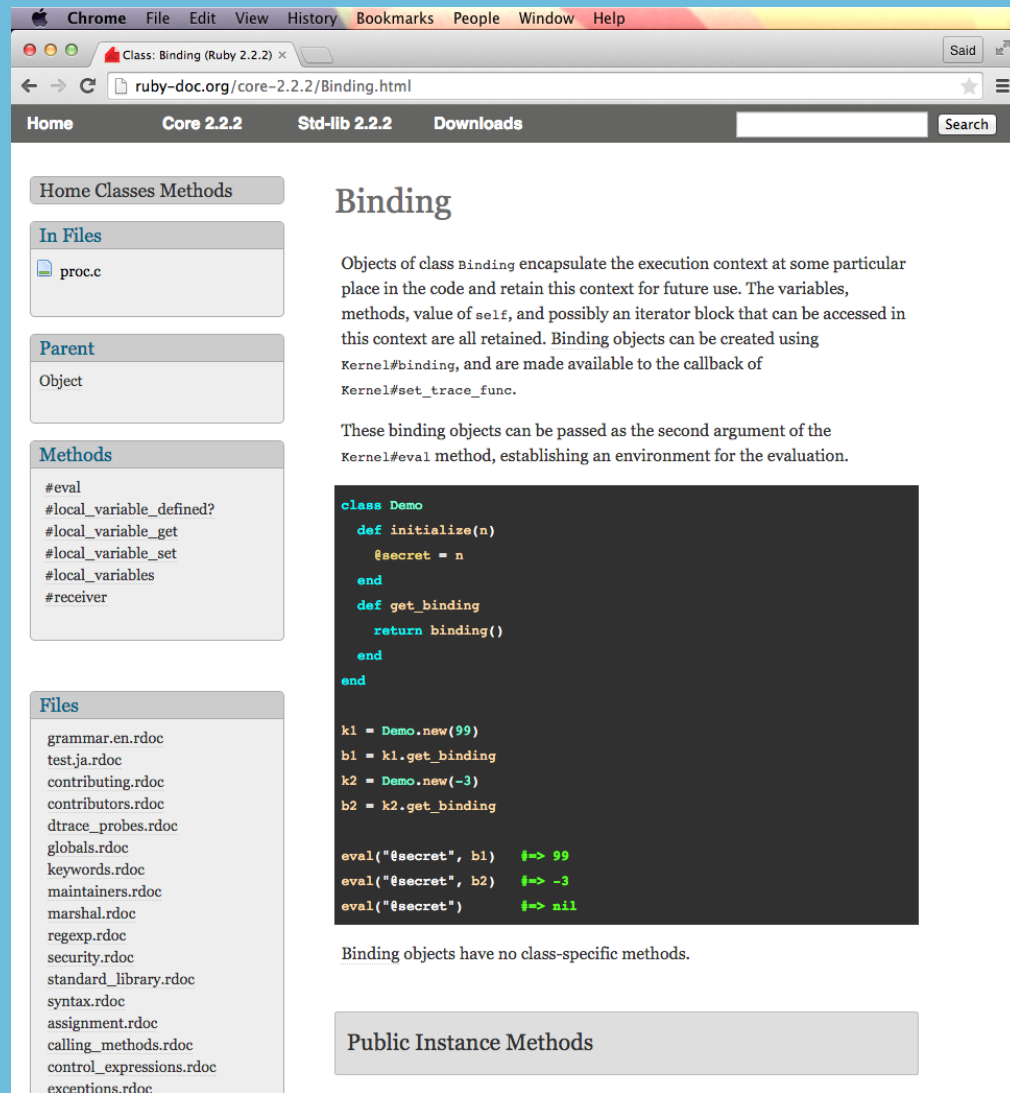
```
# From ruby documentation:  
# http://ruby-doc.org/core/Module.html#method-i-remove\_class\_variable
```

```
class Dummy  
  @@var = 99  
  puts @@var                                # 99  
  
  remove_class_variable(:@@var)  
  
  p(defined? @@var)                        # nil  
end
```

# #1 Source — Ruby Documentation

List of classes where the methods are defined:

- <http://ruby-doc.org/core/BasicObject.html>
- <http://ruby-doc.org/core/Object.html>
- <http://ruby-doc.org/core/Kernel.html>
- <http://ruby-doc.org/core/Module.html>
- <http://ruby-doc.org/core/Class.html>
- <http://ruby-doc.org/core/Binding.html>



The screenshot shows a web browser window displaying the Ruby documentation for the `Binding` class. The browser's address bar shows the URL `ruby-doc.org/core-2.2.2/Binding.html`. The page has a navigation bar with links for `Home`, `Core 2.2.2`, `Std-lib 2.2.2`, and `Downloads`. On the left side, there are several sections: `Home Classes Methods`, `In Files` (containing `proc.c`), `Parent` (containing `Object`), `Methods` (listing `#eval`, `#local_variable_defined?`, `#local_variable_get`, `#local_variable_set`, `#local_variables`, and `#receiver`), and `Files` (listing various .rdoc files like `grammar.en.rdoc`, `test.ja.rdoc`, etc.). The main content area is titled `Binding` and contains a paragraph explaining that `Binding` objects encapsulate the execution context. Below this, it states that these objects can be passed as the second argument of `Kernel#eval`. A code block shows a `Demo` class with `initialize`, `get_binding`, and `eval` methods, followed by a demonstration of creating `Demo` objects and using `eval` with their bindings. At the bottom, there is a section for `Public Instance Methods`.

Chrome File Edit View History Bookmarks People Window Help

Class: Binding (Ruby 2.2.2) × Said

← → ↺ ruby-doc.org/core-2.2.2/Binding.html

Home Core 2.2.2 Std-lib 2.2.2 Downloads Search

Home Classes Methods

In Files

proc.c

Parent

Object

Methods

#eval  
#local\_variable\_defined?  
#local\_variable\_get  
#local\_variable\_set  
#local\_variables  
#receiver

Files

grammar.en.rdoc  
test.ja.rdoc  
contributing.rdoc  
contributors.rdoc  
dtrace\_probes.rdoc  
globals.rdoc  
keywords.rdoc  
maintainers.rdoc  
marshal.rdoc  
regexp.rdoc  
security.rdoc  
standard\_library.rdoc  
syntax.rdoc  
assignment.rdoc  
calling\_methods.rdoc  
control\_expressions.rdoc  
exceptions.rdoc

## Binding

Objects of class `Binding` encapsulate the execution context at some particular place in the code and retain this context for future use. The variables, methods, value of `self`, and possibly an iterator block that can be accessed in this context are all retained. Binding objects can be created using `Kernel#binding`, and are made available to the callback of `Kernel#set_trace_func`.

These binding objects can be passed as the second argument of the `Kernel#eval` method, establishing an environment for the evaluation.

```
class Demo
  def initialize(n)
    @secret = n
  end
  def get_binding
    return binding()
  end
end

k1 = Demo.new(99)
b1 = k1.get_binding
k2 = Demo.new(-3)
b2 = k2.get_binding

eval("@secret", b1) #=> 99
eval("@secret", b2) #=> -3
eval("@secret")     #=> nil
```

Binding objects have no class-specific methods.

Public Instance Methods